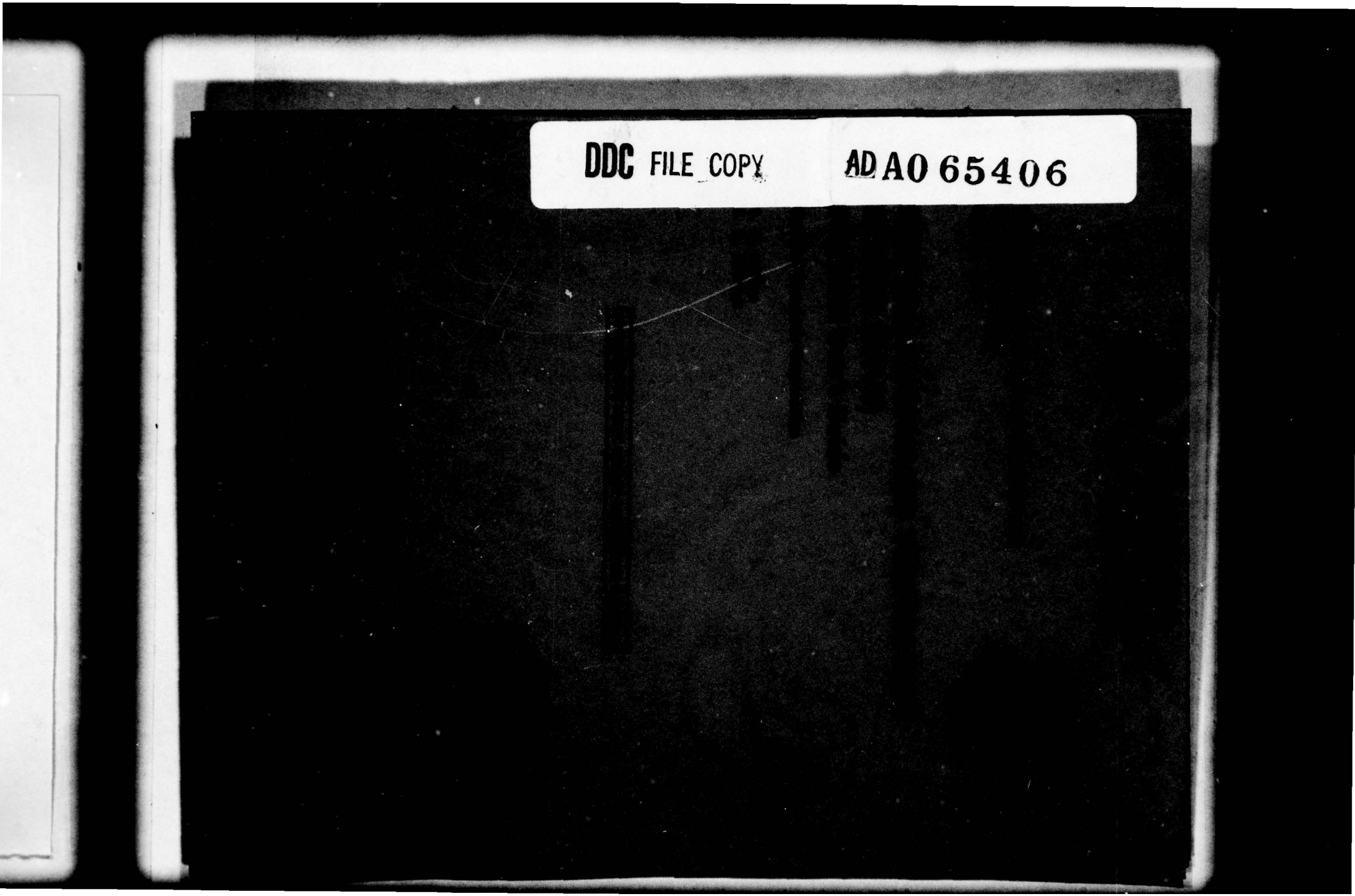
AD-A065 406 GENERAL RESEARCH CORP SANTA BARBARA CALIF F/G 9/2
FORTRAN AUTOMATED VERIFICATION SYSTEM (FAVS). VOLUME III. DMATR--ETC(U)
JAN 79 R A MELTON, D M ANDREWS F30602-76-C-0436
UNCLASSIFIED RADC-TR-78-268-VOL-3 NL

1 OF 1
ADA
065406

END
DATE
FILMED
4 79
DDC



UNCLASSIFIED (19) TR-78-268-Vol-3

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: RADC-TR-78-268, Vol III (of three)

2. GOVT ACCESSION NO.

(9) 3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): FORTRAN AUTOMATED VERIFICATION SYSTEM (FAVS): DMATRAN User's Guide, Volume III.

5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report, Oct 76 — Jan 78

6. PERFORMING ORG. REPORT NUMBER: N/A

(10) 7. AUTHOR(s): A. Melton, M. Andrews — Rich, Dorothy

(15) 8. CONTRACT OR GRANT NUMBER(s): F30602-76-C-0436

9. PERFORMING ORGANIZATION NAME AND ADDRESS: General Research Corporation, P.O. Box 6770, Santa Barbara CA 93111

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 63701B, (16) 32010320, (17) 03

11. CONTROLLING OFFICE NAME AND ADDRESS: Rome Air Development Center (ISIE), Griffiss AFB NY 13441

(11) 12. REPORT DATE: Jan 1979

13. NUMBER OF PAGES: 46

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Same — (12) 46p.

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE: N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DDC RECEIVED MAR 8 1979 B

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Frank S. Lamonica (ISIE)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Software
Software Testing
Software Verification
Software Documentation
FAVS
Automated Verification System

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

DMATRAN is a structured programming language which provides the logical constructs that are necessary to write structured code in FORTRAN. The additional control constructs in DMATRAN are ~~(1)~~ a structured IF construct which allows execution of a group of statements, ~~(2)~~ two basic structures which permit iteration while a logical expression is true (DO WHILE...END WHILE) or until a logical expression becomes true (DO UNTIL...END UNTIL), and ~~(3)~~ a non-iterative CASE structure which begins with an integer expression which is

(Cont'd) next page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Item 20 (Cont'd)

evaluated and then compared with integers in a list of CASE statements that follow. Execution of a group of statements following the matching CASE statement is then initiated. The DMATRAN language also contains an INVOKE...BLOCK... END BLOCK construct which provides a form of internal subroutine capability as well as a way to reduce overhead costs by eliminating duplicate sections of code.

This DMATRAN User's Guide describes and illustrates each structured construct. It also explains how to use the DMATRAN precompiler which translates DMATRAN into compilable FORTRAN code. The DMATRAN precompiler provides additional source text editing and display features including page ejection, suppressing source listing, and changing the syntax of DMATRAN statements; the DMATRAN commands for these capabilities are also described.

The DMATRAN precompiler has been installed on the HIS 6180 GCOS and MULTICS computer systems at the Rome Air Development Center, Griffiss AFB, New York, and on the UNIVAC 1100/42 computer systems at the Defense Mapping Agency Aerospace Center in St. Louis, Missouri, and the Defense Mapping Agency Topographic Center in Washington, D.C.

CONTENTS

SECTION		PAGE
1	INTRODUCTION	1-1
2	DMATRAN CONTROL CONSTRUCTS	2-1
	2.1 IF...THEN...ELSE...END IF	2-3
	2.2 DO WHILE...END WHILE	2-5
	2.3 DO UNTIL...END UNTIL	2-5
	2.4 CASE OF...CASE...CASE ELSE...END CASE	2-8
	2.5 BLOCK...END BLOCK AND INVOKE	2-11
3	USING THE DMATRAN PRECOMPILER	3-1
	3.1 DMATRAN Input	3-1
	3.2 DMATRAN Indented Listing	3-2
	3.3 FORTRAN Output	3-3
4	DMATRAN CONSTRAINTS	4-1
	4.1 SYNTAX	4-1
	4.2 DO UNTIL	4-1
	4.3 CASE	4-1
	4.4 BLOCK Construct	4-1
5	DISPLAY COMMANDS	5-1
	5.1 Command Form	5-1
	5.2 Source Listing	5-2
	5.3 Double-Space Around Comments	5-2
	5.4 Keyword Recognition	5-2
	5.5 Page Ejection	5-5
	5.6 Extended Comment	5-5

CONTENTS CONT.

SECTION		PAGE
APPENDIX A	SUMMARY OF DMATRAN STATEMENTS AND COMMANDS	A-1
APPENDIX B	FILE DESCRIPTIONS	B-1
APPENDIX C	JOB STREAMS	C-1
	INDEX	I-1

ILLUSTRATIONS

NO.		PAGE
1.1	DMATRAN Precompiler	1-2
2.1	DMATRAN Control Constructs	2-2
2.2	IF...THEN...END IF Construct	2-3
2.3	IF...THEN...ELSE...END IF Construct	2-4
2.4	DO WHILE...END WHILE Construct	2-6
2.5	DO UNTIL...END UNTIL	2-7
2.6	CASE OF..CASE..CASE ELSE...END CASE	2-9
2.7	DMATRAN Case Construct	2-10
2.8	BLOCK...END BLOCK and INVOKE Construct	2-12
2.9	BLOCK Cross-Reference Report	2-13
3.1	DMATRAN Source Input	3-1
3.2	DMATRAN Indented Listing	3-2
3.3	Translated FORTRAN	3-4
5.1	DMATRAN Keyword Syntax	5-3
5.2	Listing of DMATRAN Keyword Example	5-4

1 INTRODUCTION

DMATRAN is an extension to FORTRAN that simplifies structured programming by providing a convenient syntax for writing structured programming control constructs. The DMATRAN precompiler translates the DMATRAN statements into standard FORTRAN while passing all other statements unchanged to a file which can then be compiled by the FORTRAN compiler. This sequence is illustrated in Fig. 1.1. In addition to the translation, the precompiler checks the control structure for proper use of DMATRAN control structures and issues error messages if violations occur.

The precompiler provides the following additional features to improve code production:

1. Indented listing of the DMATRAN source code.
2. Editing functions which include in-line comments, double-spacing around comments, indentation control, selective page ejection, and selective suppression of the source listings.

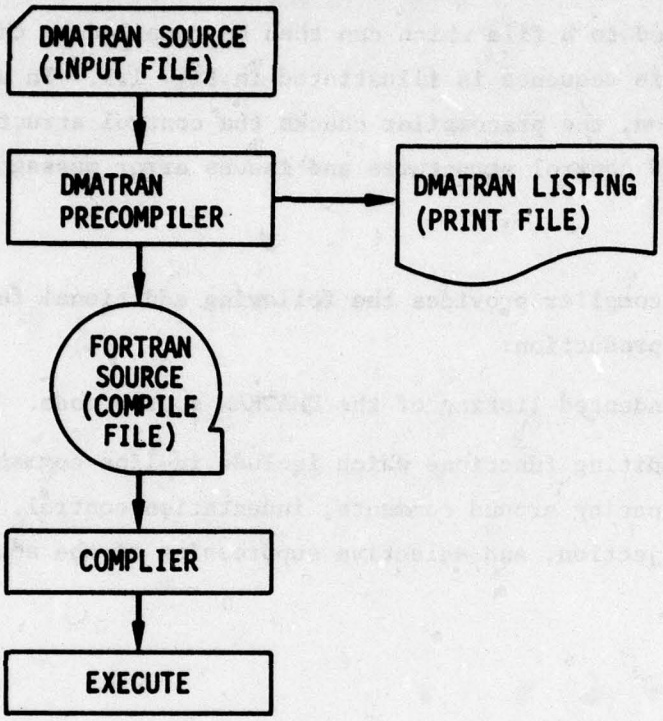


Figure 1.1. DMATRAN Precompiler

2 DMATRAN CONTROL CONSTRUCTS

DMATRAN replaces FORTRAN control statements with the following control statement constructs:

- IF...THEN...ELSE...END IF - provides block structuring of conditionally executable sequences of statements.
- DO WHILE...END WHILE - permits iteration of a code segment while a specified condition remains true.
- CASE OF...CASE...CASE ELSE...END CASE - allows multiple choices for program action selection.
- DO UNTIL...END UNTIL - permits iteration until a specified condition becomes true.
- INVOKE<name>...BLOCK<name>...END BLOCK - provides a facility for top-down programming and internal subroutines.

These statement forms can be intermixed with ordinary FORTRAN non-control statements in the text stream which is processed by the DMATRAN precompiler. DMATRAN statements are converted by the precompiler to equivalent FORTRAN statements, and the resulting file can be compiled by the FORTRAN compiler in the normal manner. A description, flowchart, and example of each DMATRAN control construct is provided in this section. The DMATRAN constructs are shown in Fig. 2.1.

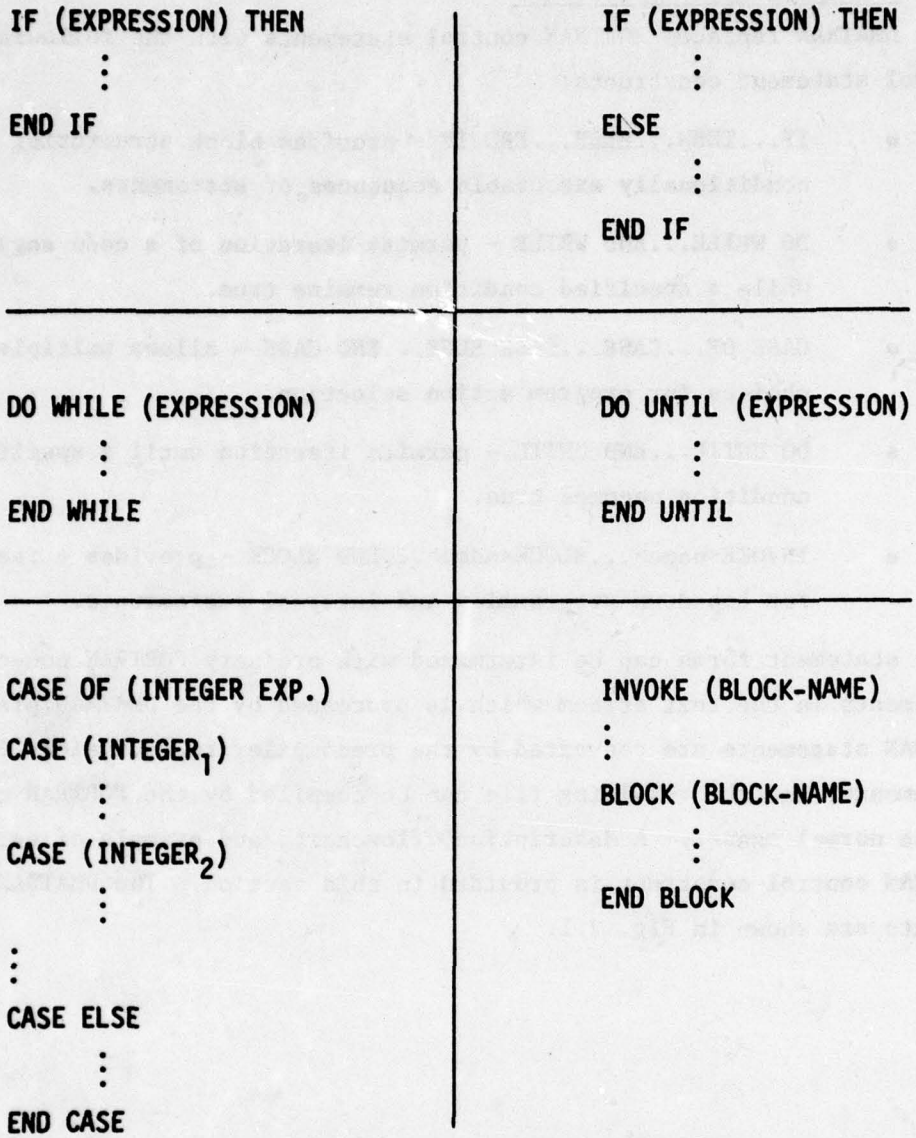


Figure 2.1. DMATRAN Control Constructs

2.1 IF...THEN...ELSE...END IF

The IF...THEN...ELSE...END IF construct provides block structuring of conditionally executable statements. The basic form of this construct is IF...THEN...END IF, illustrated in Fig. 2.2. If <expression> is true, control transfers to the first statement within the construct; otherwise, the statement immediately following the END IF will be executed. Use of the ELSE statement is optional. If the ELSE is present and <expression> is false, the statements following the ELSE are executed. This construct is illustrated in Fig. 2.3.

```
IF  (<EXPRESSION>) THEN
.
.   STATEMENTS TO EXECUTE IF <EXPRESSION> IS TRUE
.
END IF
```

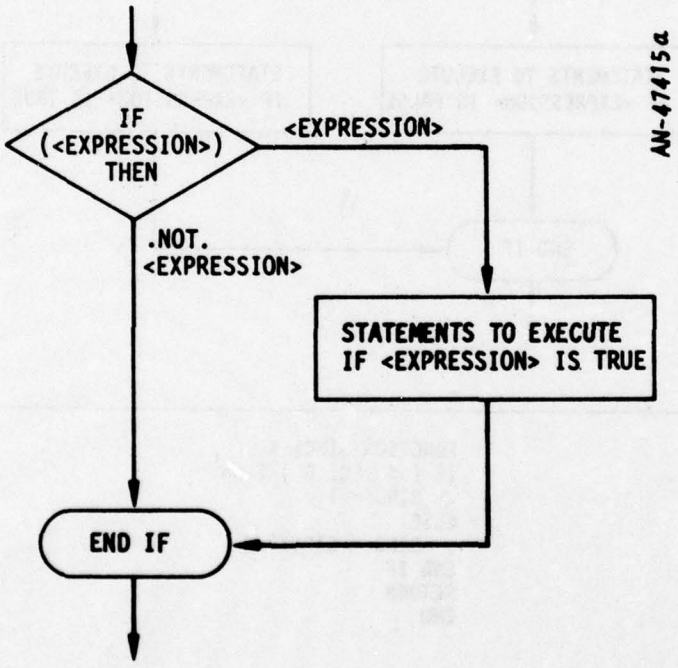
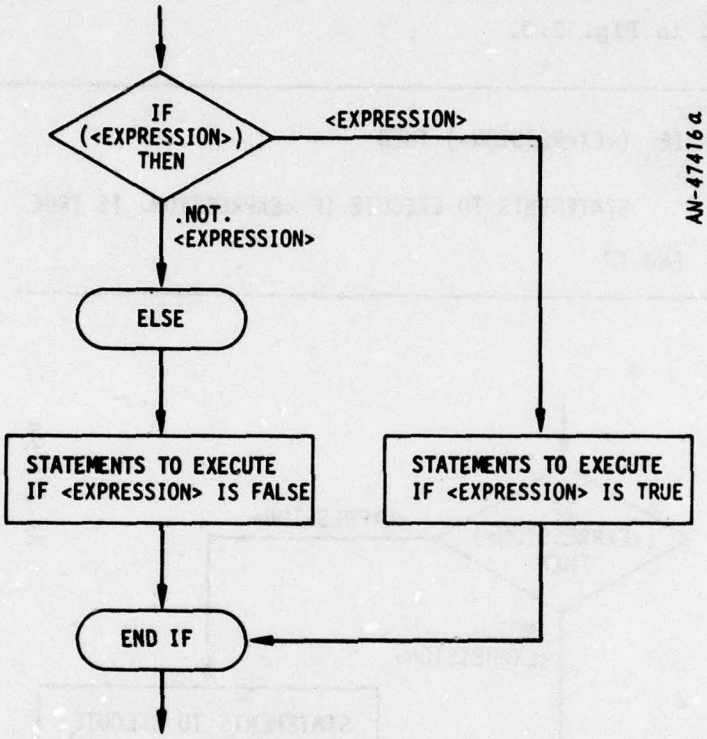


Figure 2.2. IF...THEN...END IF Construct

```
IF  (<EXPRESSION>) THEN
.
.   STATEMENTS TO EXECUTE IF <EXPRESSION> IS TRUE
.
ELSE
.
.   STATEMENTS TO EXECUTE IF <EXPRESSION> IS FALSE
.
END IF
```




```
FUNCTION SINC( X )
IF ( X .EQ. 0 ) THEN
.  SINC = 1.
ELSE
.  SINC = SIN(X)/X
END IF
RETURN
END
```

Figure 2.3. IF...THEN...ELSE...END IF Construct

2.2 DO WHILE...END WHILE

The DO WHILE...END WHILE construct indicates a repetitive operation which is to be performed zero or more times. Execution occurs in the following manner:

1. The value of <expression> is found: if true, the statements contained within the DO WHILE block are executed; if false, control passes to the statement immediately following the END WHILE.

2. If the statements within the DO WHILE block have been executed, the value of <expression> is checked again, with the same consequences as in (1).

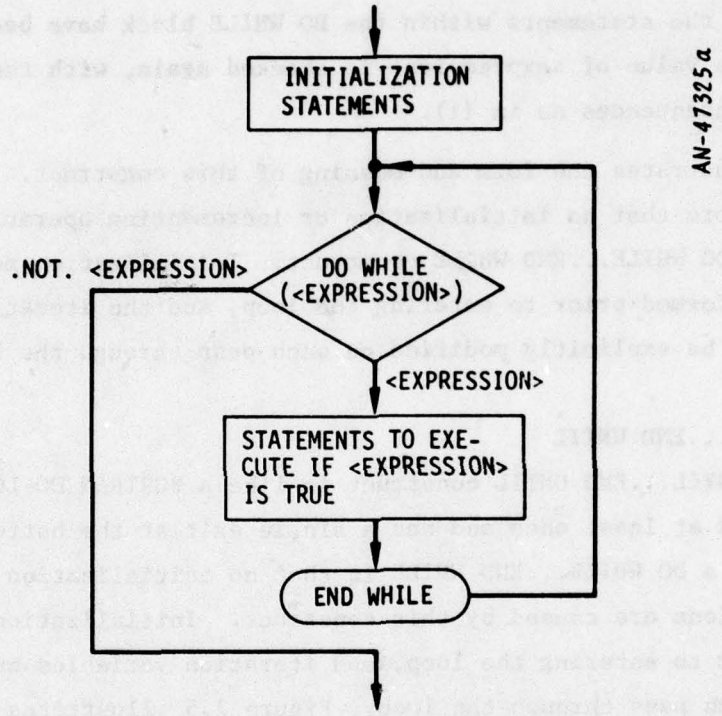
Figure 2.4 illustrates the form and meaning of this construct. It is important to note that no initialization or incrementing operations are caused by the DO WHILE...END WHILE construct. Initialization must be explicitly performed prior to entering the loop, and the iteration variables must be explicitly modified on each pass through the loop.

2.3 DO UNTIL...END UNTIL

The DO UNTIL...END UNTIL construct is like a FORTRAN DO-LOOP in that it is performed at least once and has a single exit at the bottom of the loop, and like a DO WHILE...END WHILE in that no initialization or incrementing operations are caused by this construct. Initialization must be performed prior to entering the loop, and iteration variables must be modified on each pass through the loop. Figure 2.5 illustrates this construct.

The statements enclosed within the DO UNTIL and the END UNTIL are executed at least once. Then <expression> is evaluated and, if false, iteration and evaluation of the expression continue until it is true. At that time execution of the statements following the END UNTIL begins.

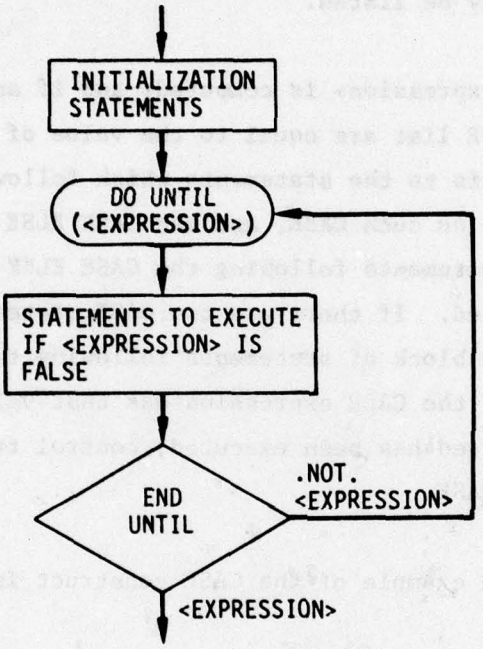
```
INITIALIZATION STATEMENTS
DO WHILE (<EXPRESSION>)
.
.      STATEMENTS TO EXECUTE IF <EXPRESSION> IS TRUE
.
END WHILE
```




```
FUNCTION SQRT( A )
X = A
DO WHILE( ABS(X-A/X) .GT. 1.E-6 )
.      X = (X+A/X)/2
END WHILE
SQRT = X
RETURN
END
```

Figure 2.4. DO WHILE...END WHILE Construct

```
INITIALIZATION STATEMENTS
DO UNTIL (<EXPRESSION>)
.
.     STATEMENTS TO EXECUTE IF <EXPRESSION> IS FALSE
.
END UNTIL
```




```
FUNCTION CONVRG(XINIT, EPS, F )
EXTERNAL F
X = XINIT
DO UNTIL (ABS(X-XOLD).LE.EPS)
.     XOLD = X
.     X = F(X)
END UNTIL
CONVRG = X
RETURN
END
```

Figure 2.5. DO UNTIL...END UNTIL

2.4 CASE OF...CASE...CASE ELSE...END CASE

The CASE statement provides a way to select which group of statements will be executed. The general form of the CASE construct consists of CASE OF...CASE...CASE ELSE...END CASE.

Figure 2.6 illustrates the CASE construct. I , J , and N represent integers of positive value. They may be in any order, and there is no limit to how many integers may be listed.

The value of <integer expression> is computed, and if any of the specified integers in the CASE list are equal to the value of <expression> then the transfer of control is to the statements which follow that particular CASE. If there is no such CASE, and the CASE ELSE statement is present, then the block of statements following the CASE ELSE is executed; otherwise, no block is executed. If there are two CASE statements with the same CASE index, then the block of statements following the first occurring one is executed (if the CASE expression has that value). After the block of statements selected has been executed, control transfers to the statement after the END CASE.

A listing containing an example of the CASE construct is shown in Fig. 2.7.

```
CASE OF (<INTEGER EXPRESSION>)
CASE (I)
.     BLOCK OF STATEMENTS
CASE (J)
.     BLOCK OF STATEMENTS
                :
CASE (N)
.     BLOCK OF STATEMENTS
CASE ELSE
.     BLOCK OF STATEMENTS
END CASE
```

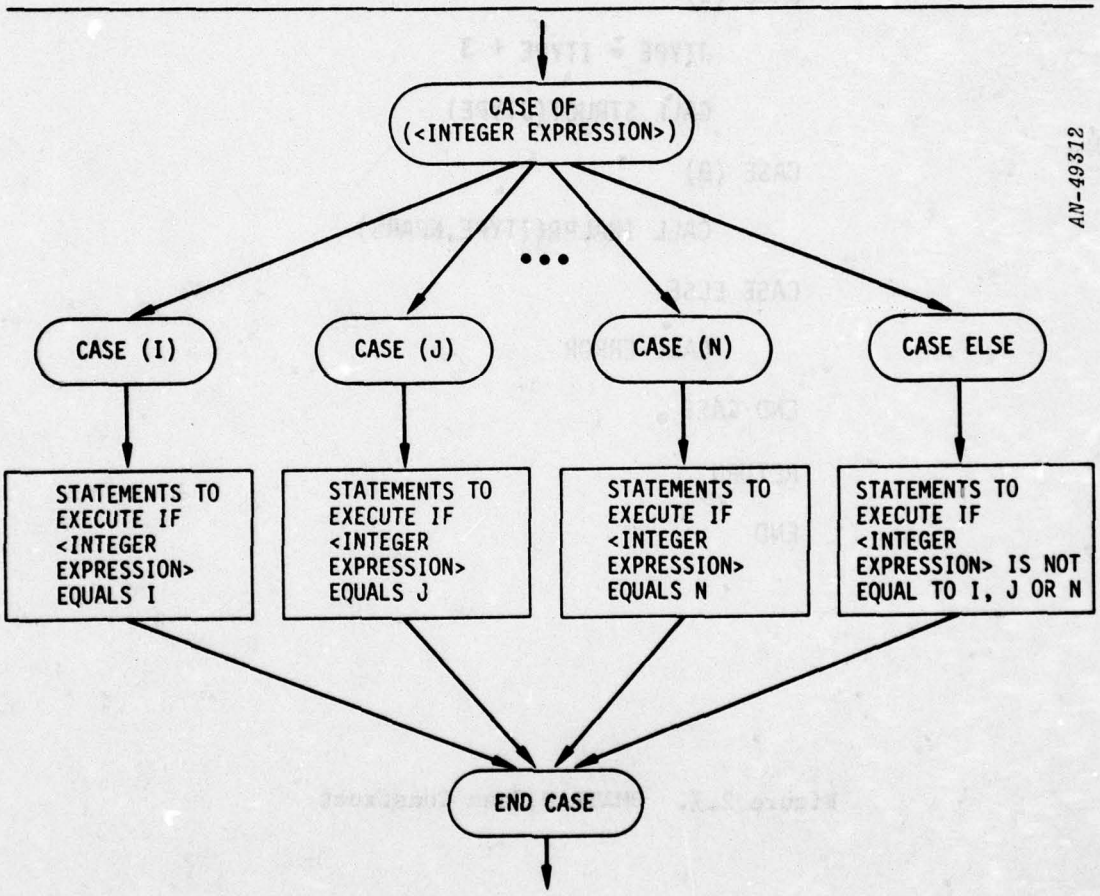


Figure 2.6. CASE OF..CASE..CASE ELSE...END CASE

```
SUBROUTINE XAMPL (ITYPE,NPARS)

:
CASE OF (ITYPE)
CASE (3)
      CALL GETCRD(ITYPE)
CASE (5)
      JTYPE = ITYPE + 3
      CALL STRUCT(JTYPE)
CASE (9)
      CALL IBALPR(ITYPE,NPARS)
CASE ELSE
      CALL ERROR
END CASE
RETURN
END
```

Figure 2.7. DMATRAN Case Construct

2.5 BLOCK...END BLOCK and INVOKE

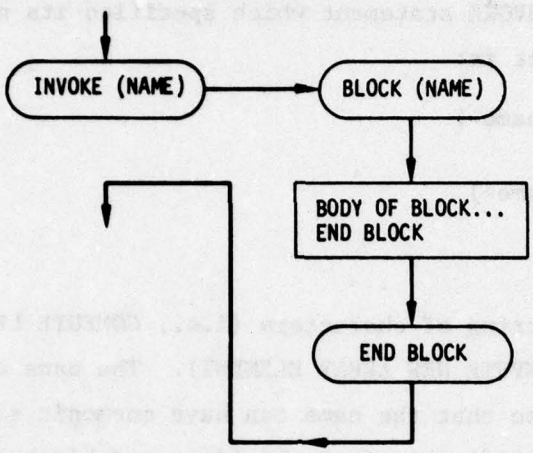
The BLOCK...END BLOCK construct provides a form of internal subroutine capability in DMATRAN source programs. This construct is an internal procedure which has access to all variables in the routine which contains it. A BLOCK...END BLOCK is executed only if it is referred to with an INVOKE statement which specifies its name. The form for this construct is:

```
INVOKE (<name>)
.
.
.
BLOCK (<name>)
      .
      .
      .
END BLOCK
```

where <name> is any string of characters (i.e., COMPUTE LENGTH, PRINT CURRENT STATUS, or COMPUTE NEW ARRAY ELEMENT). The name of a BLOCK may be arbitrarily long, so that the name can have mnemonic significance. All characters are significant after the first non-blank and before the last non-blank. The name of a BLOCK is known throughout the entire routine in which it is contained. Figure 2.8 illustrates this construct.

As the flowchart for this construct indicates, it is a single-entry (the BLOCK statement), single-exit (the END BLOCK statement) section of code. An INVOKE statement causes control to transfer to the named BLOCK statement, and the matching END BLOCK statement causes control to transfer back to the statement after the INVOKE. More than one INVOKE for a given BLOCK...END BLOCK construct is allowed. Though BLOCK...END BLOCK constructs can be nested, no recursion is allowed in the invoking of BLOCKS (i.e., a BLOCK cannot directly or indirectly invoke itself). BLOCKS cannot be invoked from an external routine, nor can they be passed as a parameter to another routine. BLOCK constructs may be placed before or after the RETURN statement.

```
INVOKE (<NAME>)
BLOCK (<NAME>)
.
.     BLOCK STATEMENTS
.
END BLOCK
```




```
SUBROUTINE MLTPLY(A,B,C,N)
DIMENSION A(10,10),B(10,10),C(10,10)
I = 1
DO WHILE ( I .LE. N )
.     J = 1
.     DO WHILE ( J .LE. N )
.     .     INVOKE ( COMPUTE NEW ARRAY ELEMENT )
.           J = J + 1
.     END WHILE
.     I = I + 1
END WHILE
BLOCK ( COMPUTE NEW ARRAY ELEMENT )
.     S = 0.0
.     K = 1
.     DO WHILE ( K .LE. N )
.     .     S = S + A(I,K) * B(K,J)
.           K = K + 1
.     END WHILE
.     C(I,J) = S
END BLOCK
RETURN
END
```

Figure 2.8. BLOCK...END BLOCK and INVOKE Construct

When the BLOCK...END BLOCK construct is used, available space for storing additional BLOCK names is indicated in the BLOCK Cross-Reference Report (Fig. 2.9) which follows the DMATRAN source listing for each module. The total number of characters in all BLOCK names used in one subroutine cannot exceed 1,000. The number of INVOKEs and the number of BLOCKs varies with each module as it is dependent on the size of the names of the BLOCKs and the number of invocations. When the maximum has been reached, a message is printed on the DMATRAN listing indicating a BLOCK name table overflow.

BLOCK CROSS-REFERENCE

BLOCK NAME	DEFINED	INVOKED			
INITIALIZE BOUNDARIES	36	27			
SORT INCOMING TABLE IN SEGMENTS	54	29			
PARTITION SORT SEGMENTED TABLE	80	30			
SORT ENTIRE TABLE	114	32			
SWITCH INCOMING TABLE BLOCKS TO SORTED	127	34			
TRANSFER FROM TABLE FOR FIRST SORT	133	59	116		
SET UP SORT BY SEGMENTS	151	83			
STORE SMALLEST IN TEMPORARY TABLE	184	90	96	110	164
MOVE FROM A SEGMENT TO ARRAY	189	92	157		
FIND WHICH SEGMENT	167	95	102	163	

645 WORDS LEFT

Figure 2.9. BLOCK Cross-Reference Report

There are many reasons for using the BLOCK construct. For example, the overhead in calling subroutines is often very high; and, in addition, variables in FORTRAN and DMATRAN programs must be passed as parameters or placed in COMMON to be accessible to both the calling routine and the subroutine which is called. Often a subroutine references only variables which are already in the calling routine. Using a BLOCK structure as a internal subroutine eliminates the need to provide a means of accessing these variables.

Another way the BLOCK construct can improve overhead costs is by elimination of duplicate sections of code. Since the same BLOCK can be invoked by more than one INVOKE statement, code can be made more efficient by putting identical sections of code into BLOCK structures. The following example illustrates the use of BLOCKs to avoid code duplication.

S_1 and S_2 in the following code represent two sets of statements. The use of a BLOCK in Method 2 below eliminates the need for duplicating code.

Method 1:

```
IF(A) THEN
      IF(C) THEN
            S1
      ELSE
            S2
      END IF
ELSE
      IF(D) THEN
            S2
      ELSE
            S1
      END IF
END IF
```

Method 2:

```
IF(A) THEN
      IF(C) THEN
            INVOKE(BLOCK-A)
      ELSE
            INVOKE(BLOCK-B)
      END IF
ELSE
      IF(D) THEN
            INVOKE(BLOCK-B)
      ELSE
            INVOKE(BLOCK-A)
      END IF
END IF
```

where the BLOCKs are defined as:

```
BLOCK(BLOCK-A)
      S1
END BLOCK
```

and

```
BLOCK(BLOCK-B)
      S2
END BLOCK
```

3 USING THE DMATRAN PREPROCESSOR

3.1 DMATRAN INPUT

Figure 3.1 illustrates a DMATRAN source program with embedded FORTRAN statements ready for input to the DMATRAN precompiler. The DMATRAN source code begins in column 7 and is not indented. More than one module may be processed in each DMATRAN run.

```
      SUBROUTINE EXAMPL (INFO,LENGTH)                                   EXAMPL1
C                                                                       EXAMPL2
C     ILLUSTRATION OF DMATRAN SYNTAX                                    EXAMPL3
C                                                                       EXAMPL4
      IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN                             EXAMPL5
      CALL CALLER ( INFO )                                              EXAMPL6
      ELSE                                                              EXAMPL7
      LENGTH=50                                                         EXAMPL8
      END IF                                                            EXAMPL9
      CASE OF (INFO+6)                                                  EXAMPL10
      CASE (14)                                                         EXAMPL11
      LENGTH=LENGTH-INFO                                                EXAMPL12
      CASE (17)                                                         EXAMPL13
      DO WHILE (INFO.LT.20)                                             EXAMPL14
      DO UNTIL (LENGTH.LE.INFO)                                         EXAMPL15
      INVOKE (COMPUTE LENGTH)                                           EXAMPL16
      IF (LENGTH.GE.30) THEN                                            EXAMPL17
      INVOKE (PRINT-RESULTS)                                            EXAMPL18
      END IF                                                            EXAMPL19
      END UNTIL                                                         EXAMPL20
      INFO=INFO+1                                                       EXAMPL21
      END WHILE                                                         EXAMPL22
      CASE ELSE                                                         EXAMPL23
      DO WHILE (LENGTH.GT.0)                                            EXAMPL24
      INVOKE (COMPUTE LENGTH)                                           EXAMPL25
      END WHILE                                                         EXAMPL26
      END CASE                                                          EXAMPL27
      BLOCK (PRINT-RESULTS)                                             EXAMPL28
      WRITE (6,1)INFO,LENGTH                                            EXAMPL29
    1 FORMAT (10X,I5,20X,I5)                                            EXAMPL30
      END BLOCK                                                         EXAMPL31
      BLOCK (COMPUTE LENGTH)                                            EXAMPL32
      LENGTH = LENGTH -10                                               EXAMPL33
      END BLOCK                                                         EXAMPL34
      RETURN                                                            EXAMPL35
      END                                                               EXAMPL36
```

Figure 3.1. DMATRAN Source Input

3.2 DMATRAN INDENTED LISTING

Figure 3.2 illustrates the automatically indented DMATRAN listing which resulted from processing the input shown in Fig. 3.1. The heading contains information from the first card of the routine being processed, as well as a page number. The leftmost column of numbers refers to the successive statements of the DMATRAN source deck. The nesting depth of each indented statement is indicated next to the statement number. Structural visibility is enhanced by connecting related DMATRAN statements with vertical dots. The dots assist in tracing paths through the program, identifying the statement number for a given line, and debugging improperly formed DMATRAN control constructs. Structural errors are indicated by error diagnostics in the DMATRAN listing. Sequence information following column 72 of the DMATRAN source cards is included on the right side of the DMATRAN listing.

```
SEQ NEST SOURCE              SUBROUTINE EXAMPL (INFO,LENGTH)                PAGE    1

  1           SUBROUTINE EXAMPL (INFO,LENGTH)                              EXAMPL1
  2      C                                                                 EXAMPL2
  3      C    ILLUSTRATION OF DMATRAN SYNTAX                               EXAMPL3
  4      C                                                                 EXAMPL4
  5           IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN                        EXAMPL5
  6  1        .   CALL CALLER ( INFO )                                     EXAMPL6
  7           ELSE                                                         EXAMPL7
  8  1        .   LENGTH=50                                                EXAMPL8
  9           END IF                                                       EXAMPL9
 10           CASE OF (INFO+6)                                             EXAMPL10
 11           CASE (14)                                                    EXAMPL11
 12  1        .   LENGTH=LENGTH-INFO                                       EXAMPL12
 13           CASE (17)                                                    EXAMPL13
 14  1        .   DO WHILE (INFO.LT.20)                                    EXAMPL14
 15  2        .   .   DO UNTIL (LENGTH.LE.INFO)                           EXAMPL15
 16  3        .   .   .   INVOKE (COMPUTE LENGTH)                         EXAMPL16
 17  3        .   .   .   IF (LENGTH.GE.30) THEN                          EXAMPL17
 18  4        .   .   .   .   INVOKE (PRINT-RESULTS)                      EXAMPL18
 19  3        .   .   .   END IF                                           EXAMPL19
 20  2        .   .   END UNTIL                                            EXAMPL20
 21  2        .   .   INFO=INFO+1                                          EXAMPL21
 22  1        .   END WHILE                                                EXAMPL22
 23           CASE ELSE                                                    EXAMPL23
 24  1        .   DO WHILE (LENGTH.GT.0)                                   EXAMPL24
 25  2        .   .   INVOKE (COMPUTE LENGTH)                             EXAMPL25
 26  1        .   END WHILE                                                EXAMPL26
 27           END CASE                                                     EXAMPL27
 28           BLOCK (PRINT-RESULTS)                                        EXAMPL28
 29  1        .   WRITE (6,1)INFO,LENGTH                                   EXAMPL29
 30  1      1 .   FORMAT (10X,I5,20X,I5)                                  EXAMPL30
 31           END BLOCK                                                    EXAMPL31
 32           BLOCK (COMPUTE LENGTH)                                       EXAMPL32
 33  1        .   LENGTH = LENGTH -10                                      EXAMPL33
 34           END BLOCK                                                    EXAMPL34
 35           RETURN                                                       EXAMPL35
 36           END                                                          EXAMPL36
```

Figure 3.2. DMATRAN Indented Listing

3.3 FORTRAN OUTPUT

Figure 3.3 illustrates a portion of the FORTRAN translation produced by the DMATRAN precompiler. The input was the SUBROUTINE EXAMPL from Fig. 3.2. The information in columns 73 to 80 of the translated FORTRAN is useful for tracing translated FORTRAN statements back to the original DMATRAN source statements. If the original source statement was a FORTRAN statement, columns 73 and 74 will contain "FO"; if it was a DMATRAN statement, column 1 is a "C" and columns 73 and 74 contain "DM." In either case columns 77 through 80 contain the original DMATRAN source statement number (the leftmost column of numbers on the DMATRAN listing) and columns 75 and 76 contain the depth of nesting of the original statement.

```
      SUBROUTINE EXAMPL (INFO,LENGTH)                                FO     1
C                                                                    FO     2
C     ILLUSTRATION OF DMATRAN SYNTAX                                 FO     3
C                                                                    FO     4
C     IF (INFO.LE.10 .AND. LENGTH.GT.0)THEN                          DM     5
      IF(INFO.LE.10 .AND. LENGTH.GT.0) GO TO 19997                          5
      GO TO 19998                                                           5
19997 CONTINUE                                                              5
      CALL CALLER ( INFO )                                           FO 1   6
C     ELSE                                                           DM     7
      GO TO 19999                                                           7
19998 CONTINUE                                                              7
      LENGTH=50                                                      FO 1   8
C     END IF                                                         DM     9
19999 CONTINUE                                                              9
C     CASE OF (INFO+6)                                               DM    10
      I9996=INFO+6                                                          10
      GO TO 19996                                                           10
C     CASE (14)                                                      DM    11
19994 CONTINUE                                                              11
      LENGTH=LENGTH-INFO                                             FO 1  12
C     CASE (17)                                                      DM    13
      GO TO 19995                                                           13
19993 CONTINUE                                                              13
C     DO WHILE (INFO.LT.20)                                          DM 1  14
19992 CONTINUE                                                              14
      IF(INFO.LT.20) GO TO 19990                                            14
      GO TO 19991                                                           14
19990 CONTINUE                                                              14
C     DO UNTIL (LENGTH.LE.INFO)                                      DM 2  15
      GO TO 19987                                                           15
19989 CONTINUE                                                              15
      IF(LENGTH.LE.INFO) GO TO 19988                                        15
19987 CONTINUE                                                              15
C     INVOKE (COMPUTE LENGTH)                                        DM 3  16
      ASSIGN 19985 TO L19985                                                16
      ASSIGN 19986 TO L19985                                                16
      GO TO 19985                                                           16
19986 CONTINUE                                                              16
C     IF (LENGTH.GE.30) THEN                                         DM 3  17
      IF(LENGTH.GE.30) GO TO 19982                                          17
      GO TO 19983                                                           17
19982 CONTINUE                                                              17
C     INVOKE (PRINT-RESULTS)                                         DM 4  18
      ASSIGN 19980 TO L19980                                                18
      ASSIGN 19981 TO L19980                                                18
      GO TO 19980                                                           18
19981 CONTINUE                                                              18
C     END IF                                                         DM 3  19
19983 CONTINUE                                                              19
19984 CONTINUE                                                              19
C     END UNTIL                                                      DM 2  20
      GO TO 19989                                                           20
19988 CONTINUE                                                              20
      INFO=INFO+1                                                    FO 2  21
C     END WHILE                                                      DM 1  22
      GO TO 19992                                                           22
19991 CONTINUE                                                              22
C     CASE ELSE                                                      DM    23
      GO TO 19995                                                           23
19979 CONTINUE                                                              23
C     DO WHILE (LENGTH.GT.0)                                         DM 1  24
19978 CONTINUE                                                              24
      .
      .
      .
```

Figure 3.3. Translated FORTRAN

4 DMATRAN CONSTRAINTS

4.1 SYNTAX

- A maximum of 20 cards per statement
- Statement labels between 10000 and 19999 should not be used because the DMATRAN preprocessor adds statement labels, beginning with label 19999 counting backwards, to the FORTRAN source code (Fig. 3.3).
- Don't transfer to labeled DMATRAN statements with FORTRAN GO TO's.
- Comments may not be interspersed within DMATRAN statements
- All two-word DMATRAN directives may be written as two separate words or merged into one; i.e., DO UNTIL or DOUNTIL.

4.2 DO UNTIL

When the DO UNTIL...END UNTIL construct is used for iteration, it is important to note that the statements contained within the construct will be executed once before the logical expression is evaluated.

4.3 CASE

The value of <integer-expression> in CASE statements must be positive.

4.4 BLOCK CONSTRUCT

- Each BLOCK...END BLOCK construct should occur after all INVOKE statements which refer to the block name, but may be before or after the RETURN statement.
- Blocks can only be entered through INVOKE statements. Sequential control transfers around BLOCK...END BLOCK constructs. Do not use a GO TO enter the middle of a BLOCK.. END BLOCK construct from outside the block.
- The maximum number of INVOKEs and BLOCKs depends on the lengths of the BLOCK names and number of invocations, see Sec. 2.5.

5 DISPLAY COMMANDS

The DMATRAN precompiler supports a variety of commands for controlling the format of DMATRAN source listings. The capabilities supported include:

- Suppressing source listing
- Double-spacing around comments
- Keyword recognition to allow a more simple DMATRAN syntax
- Page ejection
- Extended comments

5.1 COMMAND FORM

The DMATRAN command statement has two basic forms, both are FORTRAN comment statements and begin with a C in column 1. The first command statement form is

```
C<command>
```

<command> must start in column 2 and may be any of the following:

```
LIST
NOLIST
DSOK
NODS
KWOK
NOKW
EJECT
```

The second form of the DMATRAN command statement is

```
CXCOM <value>
```

<value> must be a single character in column 7; column 6 must be blank.

5.2 SOURCE LISTING

The DMATRAN source listing may be turned off and on with the two commands

```
CNOLIST
CLIST (default)
```

This feature is useful when the DMATRAN source code is large and only parts of the code are being modified.

5.3 DOUBLE-SPACE AROUND COMMENTS

Optional double-spacing around comments is obtained by the command

```
CDSOK
```

and is turned off with the command

```
CNODS (default)
```

5.4 KEYWORD RECOGNITION

The commands for control of display format are:

```
CKWOK
CNOKW (default)
```

The CKWOK command (meaning "key-word OK") allows DMATRAN statements to be written in a simpler syntax. When the DMATRAN precompiler is in keyword recognition mode, it recognizes DMATRAN statements which do not have parentheses surrounding clauses (as well as all statements normally recognized). In this mode, DMATRAN keywords are recognized if they begin a statement, contain no blanks, and are immediately followed by a blank. A problem may arise if FORTRAN is the embedded language. FORTRAN IF statements will be interpreted as DMATRAN IF statements when a blank precedes the left parenthesis of the statement. Figure 5.1 contains an example of the simpler DMATRAN syntax with English as the embedded language Figure 5.2 is the listing that results from processing the example in Fig. 5.1 which uses the DMATRAN display commands.

```
CKWOK
      SUBROUTINE TO FIND SQUARE ROOTS
CUSOK
C     FOR EACH INPUT VALUE DETERMINE THE SQUARE ROOT
C     INITIAL VALUE IS POSITIVE, REAL
      DOUNTIL  AN EOF IS ENCOUNTERED
      INVOKE  READ INPUT VALUE
      DETERMINE INITIAL ESTIMATE OF SQUARE ROOT
      DOWHILE  ESTIMATE HAS CONVERGED TO A
      IF CURRENT ESTIMATE IS WORSE THAN PREVIOUS ESTIMATE THEN
      DETERMINE NEW ESTIMATE
      END IF
      END WHILE
C     CURRENT EPSILON IS 1.0E-05
      IF ESTIMATE**2 IS WITHIN EPSILON OF A THEN
      INVOKE  PRINT SQUARE ROOT
      ELSE
      INVOKE  PRINT ERROR MESSAGE
      END IF
      END UNTIL
CNOLIST
C     THESE BLOCKS WILL BE CODED LATER
      BLOCK  READ INPUT VALUE
      END BLOCK
      BLOCK  PRINT ERROR MESSAGE
      END BLOCK
CLIST
CEJECT
CNODS
CNOKW
C     BLOCK WRITTEN 6/77
      BLOCK (PRINT SQUARE ROOT)
      PRINT 1000, ROOT
 1000 FORMAT (* ROOT = *,E12.5)
      END BLOCK
      RETURN
      END
```

Figure 5.1. DMATRAN Keyword Syntax

```
SEQ NEST SOURCE                KWOK

  1       CKWOK
  2             SUBROUTINE TO FIND SQUARE ROOTS

  3       CDSOK
  4       C     FOR EACH INPUT VALUE DETERMINE THE SQUARE ROOT
  5       C     INITIAL VALUE IS POSITIVE, REAL

  6             DOUNTIL  AN EOF IS ENCOUNTERED
  7   1         .   INVOKE  READ INPUT VALUE
  8   1         .   DETERMINE INITIAL ESTIMATE OF SQUARE ROOT
  9   1         .   DOWHILE  ESTIMATE HAS CONVERGED TO A
 10   2         .   .   IF CURRENT ESTIMATE IS WORSE THAN PREVIOUS ESTIMATE THEN
 11   3         .   .   .   DETERMINE NEW ESTIMATE
 12   2         .   .   END IF
 13   1         .   END WHILE

 14   1   C     *   CURRENT EPSILON IS 1.0E-05

 15   1         .   IF ESTIMATE**2 IS WITHIN EPSILON OF A THEN
 16   2         .   .   INVOKE  PRINT SQUARE ROOT
 17   1         .   ELSE
 18   2         .   .   INVOKE  PRINT ERROR MESSAGE
 19   1         .   END IF
 20             END UNTIL

 27       C
```

```
SEQ NEST SOURCE                KWOK

 28       C
 29       CNODS
 30       CNOKW
 31       C     BLOCK WRITTEN 6/77
 32             BLOCK (PRINT SQUARE ROOT)
 33   1         .   PRINT 1000, ROOT
 34   1    1000 .   FORMAT(* ROOT = *,E12.5)
 35             END BLOCK
 36             RETURN
 37             END
```

Figure 5.2. Listing of DMATRAN Keyword Example

5.5 PAGE EJECTION

Specification of page ejection at any point in the DMATRAN listing is especially useful in delineating BLOCK structures; it is obtained by the command

```
CEJECT
```

5.6 EXTENDED COMMENT

Comments may follow a statement by using the command

```
CXCOM <value>
```

For example, in the command

```
CXCOM ;
```

the extended comment character is considered to be ";". The statement

```
T = T + DT ; ADVANCE TIME
```

would appear on the DMATRAN listing unmodified, but the text "ADVANCE TIME" would be changed into a FORTRAN comment on the translated FORTRAN output.

```
C     ADVANCE TIME
      T = T + DT
```

APPENDIX A

SUMMARY OF DMATRAN STATEMENTS AND COMMANDS.

SUMMARY OF DMATRAN STATEMENTS AND COMMANDS

PAGE	DMATRAN STATEMENT	FUNCTION
2-3	IF...THEN...ELSE...END IF	Selection construct
2-5	DO WHILE...END WHILE	Iteration with test at top
2-5	DO UNTIL...END UNTIL	Iteration with test at bottom
2-8	CASE OF...CASE...CASE ELSE...END CASE	Selection construct
2-11	INVOKE	Execute an internal subroutine
2-11	BLOCK...END BLOCK	Internal subroutine with mnemonic name

PAGE	DMATRAN COMMAND (defaults underlined)	FUNCTION
5-2	LIST	Resume listing of DMATRAN source statements
5-2	NOLIST	Suppress listing of DMATRAN source statements
5-2	DSOK	Double-space around comments
5-2	NODS	Suppress double-spacing around comments
5-2	KWOK	Enter keyword recognition mode
5-2	NOKW	Leave key word recognition node
5-5	EJECT	New page
5-5	XCOM <char>	Set the extended comment character

APPENDIX B

FILE DESCRIPTIONS

FILES USED AT RADC INSTALLATION

UNIT	FILE NAME	DESCRIPTION
1	INPUT	DMATRAN source input
2	PRINT	Indented listing of DMATRAN source
3	COMPILE	Compilable FORTRAN output

FILES USED AT DMA INSTALLATIONS

UNIT	FILE NAME	DESCRIPTION
5	INPUT	DMATRAN source input
6	PRINT	Indented listing of DMATRAN source
3	COMPILE	Compilable FORTRAN output

APPENDIX C

JOB STREAMS

RADC HONEYWELL 6180/GCOS

SAMPLE DMATRAN JOB STREAM

The job stream in the following example can be used to execute the DMATRAN precompiler.

```
1.   $    IDENT
2.   $    SELECT   BFCBGRC4/DMATRAN/EXECUTE
3.   $    PRMFL    01,R,S,(BCD dmatran source file)
4.   $    PRMFL    03,W,S, (BCD translated FORTRAN source file)
5.   $    ENDJOB
```

The BCD DMATRAN Source File may have been generated by a programmer or by the FAVS Restructure Option (See FAVS User's Guide, General Research Corporation CR-1-754).

RADC HONEYWELL 6180/MULTICS

SAMPLE DMATRAN JOB STREAM

(USING THE GCOS ENCAPSULATOR)

In order to use the DMATRAN precompiler, using source code written in DMATRAN generated by a programmer or by FAVS restructurer, the job stream shown in the following example can be used.

```
1.  $   snumb   (number)
2.  $   ident
3.  $   program rlhs
4.  $   limits  (CP time limit),32k,,(print line limit)
5.  $   prmfl   h*,r,r,>udd>3201c0320>Urban>dmatran>hstar
6.  $   select  >udd>3201c0320>Urban>dmatran>filedefs -ascii
7.  $   prmfl   01,r,s,>udd>(BCD dmatran source file)
8.  $   prmfl   03,w,s,>udd>(BCD Translated FORTRAN source file)
9.  $   endjob
```

DMA UNIVAC 1100/42
SAMPLE DMATRAN JOB STREAM

The job stream in the following example can be used to execute the DMATRAN precompiler.

```
@ASG,A      YOURSOURCE.            .YOUR DMATRAN SOURCE
@USE        Y.,YOURSOURCE.         .
@ASG,A      DBM*FAVS-DMA.          .ASG DMATRAN PRECOMPILER
@USE        DMA.,DBM*FAVS-DMA.     .
@XQT        DMA.TRAN               .EXECUTE DMATRAN PRECOMPILER
@ADD        Y.ELEMENTS             .ADD DMATRAN SOURCE ELEMENTS HERE
```

The UNIVAC 1100/42 installation of the DMATRAN precompiler supports an additional command (see Sec. 5.1) to assist in compiling translated DMATRAN. This command contains CFOR in columns 1 thru 4, followed by any desired information in columns 5 thru 80. The DMATRAN precompiler changes the C in column 1 of all CFOR commands to an @ character as the CFOR command is written to the FORTRAN output file. When the DMATRAN precompiler automatically adds the FORTRAN output file to the runstream, the translated CFOR cards direct the FORTRAN V compiler. Note that to compile a DMATRAN source element, the first line in the element should be a CFOR command. Indented listings without FORTRAN V compilations may be obtained by omitting CFOR commands.

INDEX

	PAGES
Automatic Indentation	3-2
BLOCK Construct	2-11, 4-1
BLOCK, Examples	2-12, 2-15
BLOCKs, Maximum Number of	2-13
BLOCK Name	2-11, 2-13
Card, Maximum Number per Statement	4-1
CASE Construct	2-8, 4-1
CASE, Examples of	2-9, 2-10
COMMENTs	5-1, 5-2
Depth, Nesting	3-2, 3-3
Display Commands	5-1
DMA Files	B-2
DMA Job Streams	C-4
DMATRAN	
Commands	5-1
Constructs	2-1
Guidelines	4-1
Input	3-1
Output	3-2, 3-3
DO UNTIL Construct	2-5, 4-1
DO UNTIL, Examples of	2-7
DO WHILE Construct	2-5
DO WHILE, Examples of	2-6
DSOK Command	5-2, 5-3, 5-4
ELSE	2-3
EJECT Command	5-5
Files	
DMA	B-2
RADC	B-2
IF...THEN construct	2-3
IF( )THEN, Examples of	2-3, 2-4
INVOKE, Examples of	2-12, 2-15
INVOKE, Number of	2-13
INVOKE Statement	2-11, 4-1
JOB Streams	
DMA UNIVAC 1100/42	C-4
RADC HONEYWELL 6180/GCOS	C-2
RADC HONEYWELL 6180/MULTICS	C-3

	PAGES
KWOK Command	5-2, 5-3, 5-4
Label, Initial Generation of	4-1
Level, Nesting	3-2, 3-3
LIST Command	5-2, 5-3, 5-4
Listing, FORTRAN	3-3
Listing, DMATRAN	3-2
Listing Unit, DMATRAN	B-2
Nesting Depth	3-2, 3-3
NODS Command	5-2, 5-3, 5-4
NOKW Command	5-2, 5-3, 5-4
NOLIST Command	5-2, 5-3, 5-4
Number of BLOCKs	2-13
Number of INVOKEs	2-13
RADC Files	B-2
RADC Job Streams	C-2, C-3
Structured Programming	1-1
XCOM Command	5-5

DMATRAN CONTROL CONSTRUCTS

CONDITIONAL

```
IF (logical expression) THEN
  :
[ELSE]
  :
END IF
```

```
CASE OF (integer expression)
{CASE (index {, index})}
  :
[CASE ELSE]
  :
END CASE
```

LOOP

```
DO WHILE (logical expression)
  :
END WHILE
```

```
DO UNTIL (logical expression)
  :
END UNTIL
```

SEQUENTIAL

```
INVOKE (block name)
  :
BLOCK (block name)
  :
END BLOCK
```

[ ] = optional

{ } = optional an arbitrary number of times

DMATRAN COMMANDS (Defaults Underlined)

LIST	List source statements
NOLIST	Suppress listing of source statements
DSOK	Double-space around comments
NDOS	Single-space around comments
KWOK	Reserved keywords identify DMATRAN statements
NOKW	No reserved keywords
EJECT	New page
XCOM <char>	Set the extended comment character